UNLocBox: short user guide

Matlab convex optimization toolbox

Lausanne - July 2012

Perraudin Nathanaël, Shuman David
Vandergheynst Pierre and **Puy Gilles**

LTS2 - EPFL

Contents

1 Introduction **1**

2 Installation and initialisation **1**

3 Solver **1**
3.1 Forward backward algorithm 1
3.2 Douglas-Rachford 2

4 The Douglas-Rachford Algorithm **2**
4.1 Alternating-direction method of multipliers (ADMM) 3
4.2 Forward backward algorithm 4
4.3 Generalized Douglas-Rachford or parallel proximal algorithm (PPXA) 4
4.4 Simultaneous-direction method of multipliers (SDMM) 5
4.5 General optional parameter for solvers 5

5 Proximal operators **6**
5.1 Presentation of different norms 6
5.2 Norm proximal operators 6
5.2.1 ℓ_1 norm proximal operator 7
5.2.2 ℓ_{12} norm proximal operator 7
5.2.3 $\ell_{1\infty}$ norm proximal operator 8
5.2.4 TV norm proximal operator 8
5.2.5 3D TV norm proximal operator 8
5.2.6 Nuclear norm proximal operator 9
5.3 Projection operator 9
5.3.1 Projection on B1-ball 9
5.3.2 Projection on B2-ball 9
5.4 Sum of proximal operator 10

6 Examples **10**
6.1 Example 1: inpainting 10
6.2 Example 2: denoising 11

7 Summary of all functions **13**

References **16**

1 Introduction

This toolbox is designed to solve convex optimization problems of the form

$$\min_{x\in\mathbb{R}^N} f_1(x) + f_2(x), \tag{1}$$

or more generally

$$\min_{x\in\mathbb{R}^N} \sum_{n=1}^{K} f_n(x), \tag{2}$$

where the f_i are lower semi-continuous convex functions from $\mathbb{R}^N$ to $(-\infty, +\infty]$. We assume $\lim_{\|x\|_2\to\infty} \left\{\sum_{n=1}^K f_n(x)\right\} = \infty$ and the f_i have non-empty domains, where the domain of a function f is given by

$$\mathrm{dom} f := \{x \in \mathbb{R}^n : f(x) < +\infty\}.$$

In problem (2), and when both f_1 and f_2 are smooth functions, gradient descent methods can be used to solve (1); however, gradient descent methods cannot be used to solve (1) when f_1 and/or f_2 are not smooth. In order to solve such problems more generally, we implement several algorithms including the forward-backward algorithm [1]- [3] and the Douglas-Rachford algorithm [4]- [8]. [1]

Both the forward-backward and Douglas-Rachford algorithms fall into the class of *proximal splitting algorithms*. The term *proximal* refers to their use of proximity operators, which are generalizations of convex projection operators. The *proximity operator* of a lower semi-continuous convex function $f : \mathbb{R}^N \to \mathbb{R}$ is defined by

$$\mathrm{prox}_f(x) := \arg\min_{y\in\mathbb{R}^N} \left\{ \frac{1}{2}\|x - y\|_2^2 + f(y) \right\}. \tag{3}$$

Note that the minimization problem in (3) has a unique solution for every $x \in \mathbb{R}^N$, so $\mathrm{prox}_f : \mathbb{R}^N \to \mathbb{R}^N$ is well-defined. The proximity operator is a useful tool because (see, e.g., [9, 10]) x^* is a minimizer in (1) if and only if for any $\gamma > 0$,

$$x^* = \mathrm{prox}_{\gamma(f_1+f_2)}(x^*). \tag{4}$$

The term *splitting* refers to the fact that the proximal splitting algorithms do not directly evaluate the proximity operator $\mathrm{prox}_{\gamma(f_1+f_2)}(x)$, but rather try to find a solution to (4) through sequences of computations involving the proximity operators $\mathrm{prox}_{\gamma f_1}(x)$ and $\mathrm{prox}_{\gamma f_2}(x)$ separately. The recent survey [11] provides an excellent review of proximal splitting algorithms used to solve (1) and related convex optimization problems.

2 Installation and initialisation

3 Solver

3.1 Forward backward algorithm

The forward-backward algorithm can be used to solve

$$\min_{x\in\mathbb{R}^N} f_1(x) + f_2(x),$$

when either f_1 or f_2 is a continuously differentiable convex function with a Lipschitz continuous gradient. A function f has a β-Lipschitz-continuous gradient ∇f if

$$\|\nabla f(x) - \nabla f(x)\|_2 \leqslant \beta\|x - y\|_2 \qquad \forall x, y \in \mathbb{R}^N, \tag{5}$$

1. In fact, the toolbox implements generalized versions of these algorithms that can solve problems with sums of a general number of such functions, but for simplicity, we discuss here the simplest case of two functions.

where $\beta > 0$.

If, without loss of generality, f_2 is the function with a β-Lipschitz-continuous gradient ∇f_2, then x^* is a solution to (1) if and only if for any $\gamma > 0$ (see, e.g., [3, Proposition 3.1]),

$$x^* = \text{prox}_{\gamma f_1}\left(x^* - \gamma \nabla f_2(x^*)\right). \tag{6}$$

The forward-backward algorithm finds a point satisfying (6) by computing a sequence $\left\{x^{(k)}\right\}_{k=0,1,\ldots}$ via

$$x^{(k+1)} = \text{prox}_{\gamma f_1}\left(x^{(k)} - \gamma \nabla f_2(x^{(k)})\right). \tag{7}$$

For any $x^{(0)}$, the sequence $\left\{x^{(k)}\right\}_{k=0,1,\ldots}$ converges to a point satisfying (6), which is therefore a minimizer of (1). For a detailed convergence analysis that includes generalizations of (7) which may result in improved convergence rates, see [3, Theorem 3.4].

The associated Matlab function `forward_backward` takes four parameters

`function sol = forward_backward (x0, f1, f2, param).`

`x0` $\in \mathbb{R}^N$ is the starting point for the algorithm. `f1` and `f2` are two Matlab structures that represent the functions f_1 and f_2. Each of these structures needs at least two fields. The Matlab structure `f1` contains the fields `f1.norm` and `f1.prox`. The former is a Matlab function that takes as input a vector $x \in \mathbb{R}^N$ and returns the value $f_1(x)$, the latter is a Matlab function that takes as input a vector $x \in \mathbb{R}^N$, a strictly positiv real number τ and returns the vector $\text{prox}_{\tau f_1}(x)$ (In Matlab, write: `f1.prox=@(x, T) prox_f1(x, T)`, where `prox_f1(x, T)` solves the problem $\text{prox}_{Tf_1}(x)$ given in equation 3). In the same way, the Matlab structure `f2` contains the fields `f2.norm` and `f2.grad`. The former is a Matlab function that takes as input a vector $x \in \mathbb{R}^N$ and returns the value $f_2(x)$, the latter is also a Matlab function that takes as input a vector $x \in \mathbb{R}^N$ and returns the vector $\nabla f_2(x)$ (In Matlab, write: `f2.grad=@(x) grad_f2(x)`, where `grad_f2(x)` return the value of $\nabla f_2(x)$). Finally, `param` is a Matlab structure that containing a set of optional parameters. The list of parameters is described in Section 4.5. The following two fields are specific to the `forward_backward` function:

- `param.method`: "ISTA" or "FISTA". Specify the method used to solve problem (1). ISTA stands for the original forward-backward algorithm while FISTA stands for its accelerated version (for details, see [12]).
- `param.gamma`: step-size parameter γ. This constant should satisfy: $\gamma \in [\epsilon, 2/\beta - \epsilon]$, for $\epsilon \in]0, \min\{1, 1/\beta\}[$.
- `param.lambda`:λ weight of the update term used in ISTA method $\in [\epsilon, 1]$. By default, it's equal to one.

3.2 Douglas-Rachford

4 The Douglas-Rachford Algorithm

The Douglas-Rachford algorithm is a more general algorithm to solve

$$\min_{x \in \mathbb{R}^N} f_1(x) + f_2(x)$$

that does not require any assumptions on the smoothness of f_1 or f_2. For any constant $\gamma > 0$, a point $x^* \in \mathbb{R}^N$ is a solution to (1) if and only if there exists a $y^* \in \mathbb{R}^N$ such that [8, Proposition 18]

$$x^* = \text{prox}_{\gamma f_2}(y^*), \text{ and} \tag{8}$$

$$\text{prox}_{\gamma f_2}(y^*) = \text{prox}_{\gamma f_1}\left(2\text{prox}_{\gamma f_2}(y^*) - y^*\right). \tag{9}$$

To find x^* and y^* that satisfy (8) and (9), the Douglas-Rachford algorithm computes the following sequence of points, for any fixed $\gamma > 0$ and stepsize $\lambda \in (0, 2)$:

$$x^{(k)} = \text{prox}_{\gamma f_2}(y^{(k)}), \text{ and} \tag{10}$$

$$y^{(k+1)} = y^{(k)} + \lambda\left(\text{prox}_{\gamma f_1}\left(2x^{(k)} - y^{(k)}\right) - x^{(k)}\right). \tag{11}$$

The convergence of the Douglas-Rachford algorithm is analyzed in [7, Corollary 5.2] and [8, Theorem 20], which also show that the algorithm can be accelerated by allowing the step size λ to change at every iteration of the algorithm. Note that while the Douglas-Rachford algorithm does not require any smoothness assumptions on f_1 or f_2, it requires two proximal steps at each iteration, as compared to one proximal step per iteration for the forward-backward algorithm.

The associated Matlab function `douglas_rachford` takes four parameters

```
function sol = douglas_rachford (x0, f1, f2, param).
```

As in Section 3.1, `x0` $\in \mathbb{R}^N$ is the starting point for the algorithm. `f1` and `f2` are two Matlab structures that represent the functions f_1 and f_2. The Matlab structure `f1` contains the fields `f1.norm` and `f1.prox`. The former is a Matlab function that takes as input a vector $x \in \mathbb{R}^N$ and returns the value $f_1(x)$, the latter is a Matlab function that takes as input a vector $x \in \mathbb{R}^N$, a strictly positiv real number τ and returns the vector $\text{prox}_{\tau f_1}(x)$. The Matlab structure `f2` contains the exact same fields but for the function f_2. Finally, `param` contains a list of parameters. The list of parameters is described in Section 4.5. The following two fields are specific to the `douglas_rachford` function:

- `param.lambda`: λ acts as a stepsize parameter. Let $\epsilon \in]0, 1[$, λ should be in the interval $[\epsilon, 2 - \epsilon]$. Its default value is 1.
- `param.gamma`: $\gamma > 0$ controls the speed of convergence. Its default value is 1.

4.1 Alternating-direction method of multipliers (ADMM)

Augmented Lagrangian techniques are classical approaches for solving problem like:

$$\min_{x \in \mathbb{R}^N} f_1(x) + f_2(Lx). \tag{12}$$

First we reformulate (12) to

$$\min_{\substack{x \in \mathbb{R}^N, y \in \mathbb{R}^M \\ Lx = y}} f_1(x) + f_2(y).$$

We then solve this problem using the augmented Lagrangian technique.

Warning: the proximal operator of function f_1 is defined to be:

$$\text{prox}^L_{f_1 \tau}(z) = \arg\min_{x \in \mathbb{R}^N} \tau f_1(x) + \frac{1}{2}\|Lx - z\|_2^2$$

The ADMM algorithm can be used when f_1 and f_2 are in $\Gamma_0(\mathbb{R}^N)$ and in $\Gamma_0(\mathbb{R}^M)$ with $L^T L$ invertible and $(\text{ridom} f_1) \cap L(\text{ridom} f_2) \neq \emptyset$. The associated Matlab function `ADMM` takes five parameters:

```
function sol = admm(x_0, f1, f2, L, param).
```

As in Section 3.1, `x0` $\in \mathbb{R}^N$ is the starting point for the algorithm. `f1` and `f2` are two Matlab structures that represent the functions f_1 and f_2. The Matlab structure `f1` contains the fields `f1.norm` and `f1.prox`. The former is a Matlab function that takes as input a vector $x \in \mathbb{R}^N$ and returns the value $f_1(x)$, the latter is a Matlab function that takes as input a vector $x \in \mathbb{R}^N$, a strictly positiv real number τ and returns the vector $\text{prox}_{\tau f_1}(x)$. The Matlab structure `f2` contains the exact same fields but for the function f_2. Finally, `param` contains a list of parameters. The list of parameters is described in Section 4.5. The following two fields are specific to the `douglas_rachford` function:

- `param.gamma`: $\gamma > 0$ controls the speed of convergence. Its default value is 1.

4.2 Forward backward algorithm

The forward backward algorithm can be used to minimize K functions problems of the form

$$\min_{x \in \mathbb{R}^N} \sum_{n=1}^{K} f_n(x),$$

with $\{f_n\}_{n \in \{1,2,...K\}} \in \Gamma_0(\mathbb{R}^N)$ and with at least one continuously differentiable convex function denoted f_1 with β-Lipschitz continuous gradient ∇f_1.

$$\|\nabla f_1(x) - \nabla f_1(x)\|_2 \leqslant \beta \|x - y\|_2 \quad \forall (x, y) \in \mathbb{R}^N \times \mathbb{R}^N,$$

where $\beta > 0$.

The associated Matlab function `generalized_forward_backward` takes four parameters

```
function sol = generalized_forward_backward(x_0, F, f1 , param).
```

`x0` $\in \mathbb{R}^N$ is the starting point for the algorithm. `f1` is a Matlab structures that represent the functions f_1. It contains the fields `f1.norm` and `f1.grad`. The former is a Matlab function that takes as input a vector $x \in \mathbb{R}^N$ and returns the value $f_1(x)$, the latter is also a Matlab function that takes as input a vector $x \in \mathbb{R}^N$ and returns the vector $\nabla f_2(x)$ (In Matlab, write: `f2.grad=@(x) grad_f2(x)`, where `grad_f2(x)` return the value of $\nabla f_2(x)$). `F` is an array of structures representing the functions $f_2, f_3, ..., f_K$. Each of these structures needs at least two fields: `F(i).norm` and `F(i).prox` with $i \in \{1, 2, ..., K-1\}$. The former is a Matlab function that takes as input a vector $x \in \mathbb{R}^N$ and returns the value $f_{i+1}(x)$, the latter is a Matlab function that takes as input a vector $\in \mathbb{R}^N$, a strictly positiv real number τ and returns the vector $\text{prox}_{\tau f_{i+1}}(x)$ (In Matlab, write: `F(i).prox=@(x, T) prox_fi(x, T)`, where `prox_fi(x, T)` solves the problem $\text{prox}_{Tf_{i+1}}(x)$ given in equation 3).

Finally, `param` is a Matlab structure that containing a set of optional parameters. The list of parameters is described in Section 4.5. The following two fields are specific to the `generalized_forward_backward` function:

- `param.gamma`: step-size parameter γ. This constant should satisfy: $\gamma \in [\epsilon, 2/\beta - \epsilon]$, for $\epsilon \in]0, \min\{1, 1/\beta\}[$.
- param.lambda: λ weight of the update term $\in [\epsilon, 1]$. By default, it's equal to one.

4.3 Generalized Douglas-Rachford or parallel proximal algorithm (PPXA)

The PPXA algorithm can be used to minimize K functions problem of the form

$$\min_{x \in \mathbb{R}^N} \sum_{n=1}^{K} f_n(x),$$

with $\{f_n\}_{n \in \{1,2,...K\}} \in \Gamma_0(\mathbb{R}^N)$. The associated Matlab function `ppxa` takes three parameters

```
function sol = ppxa(x0,F,param).
```

`x0` $\in \mathbb{R}^N$ is the starting point for the algorithm. Then, `F` is an array of structures representing the functions $f_1, f_2, ..., f_K$. Each of these structures needs at least two fields: `F(i).norm` and `F(i).prox` with $i \in \{1, 2, ..., K\}$. The former is a Matlab function that takes as input a vector $x \in \mathbb{R}^N$ and returns the value $f_i(x)$, the latter is a Matlab function that takes as input a vector $\in \mathbb{R}^N$, a strictly positiv real number τ and returns the vector $\text{prox}_{\tau f_i}(x)$ (In Matlab, write: `F(i).prox=@(x, T) prox_fi(x, T)`, where `prox_fi(x, T)` solves the problem $\text{prox}_{Tf_i}(x)$ given in equation 3). Finally, `param` contains a list of parameters. The list of parameters is described in Section 4.5. The following two fields are specific to the `ppxa` function:

- `param.W`: - W the weight (all equal by default). Note that $\sum_{i=1}^{K} \text{param.W}(i) = 1$.
- `param.gamma`: step-size parameter γ. This constant should satisfy: $\gamma \in [\epsilon, 2/\beta - \epsilon]$, for $\epsilon \in]0, \min\{1, 1/\beta\}[$.
- param.lambda: λ weight of the update term used $\in [\epsilon, 1]$. By default, it's equal to one.

4.4 Simultaneous-direction method of multipliers (SDMM)

The SDMM algorithm can be used to minimize K functions problem of the form

$$\min_{x \in \mathbb{R}^N} \sum_{n=1}^{K} f_n(L_n x),$$

with $\{f_n\}_{n\in\{1,2,...K\}} \in \Gamma_0(\mathbb{R}^N)$ and $\{L_n\}_{n\in\{1,2,...K\}}$, linear operators such that $\sum_{n=1}^{K} L_n^t L_n$ is invertible. The associated Matlab function `SDMM` takes two parameters

```
sol = sdmm(F, param)
```

`F` is an array of structures representing the functions $f_1, f_2, ..., f_K$. Each of these structures needs at least three fields: `F(i).norm`, `F(i).prox` and `F(i).x0` with $i \in \{1, 2, ..., K\}$. The first is a Matlab function that takes as input a vector $x \in \mathbb{R}^N$ and returns the value $f_i(x)$, the second is a Matlab function that takes as input a vector $\in \mathbb{R}^N$, a strictly positiv real number τ and returns the vector $\text{prox}_{\tau f_i}(x)$ (In Matlab, write: `F(i).prox=@(x, T) prox_fi(x, T)`, where `prox_fi(x, T)` solves the problem $\text{prox}_{Tf_i}(x)$ given in equation 3) and the last, `F(i).x0` is the vector used as starting point. Additionally, the linear operator L_i can be defined in a matrix form:`F(i).L`. By default the identity matrix is chosen. Finally, `param` contains a list of parameters. The list of parameters is described in Section 4.5. The following two fields are specific to the `sdmm` function:

– `param.gamma`: step-size parameter γ. This constant should satisfy: $\gamma \in [\epsilon, 2/\beta - \epsilon]$, for $\epsilon \in]0, \min\{1, 1/\beta\}[$.

4.5 General optional parameter for solvers

The optional parameter param can be used to tune most of the functions in the toolbox (see section 7). The table 4.5 summarizes the principal optional parameters

Parameter	**Explanation**	**Default value**
`param.epsilon`	This parameter is used to define the stopping criterion of the problem. The algorithm stops if $\frac{f(t) - f(t-1)}{f(t)} < \epsilon$ where $f(t)$ is the function to be minimized at iteration t. $\epsilon \in \mathbb{R}_+^*$	10^{-2}
`param.max_iter`	The maximum number of iterations	200
`param.verbose`	Log parameter: 0 no log, 1 a summary at convergence, 2 print main steps	1

Table 1: Optional parameter for solvers

5 Proximal operators

5.1 Presentation of different norms

Usual proximal operator minimize norm. In the following table, we present the main of them. For a continuous signal $x(t)$ defined on a open set $\Omega \subset \mathbb{R}^p$ we define different norms in table 2. For a discrete signal $x(n)$ with $n \in 1, 2, ..., N$, we also define norms in table 2.

Name **Notation** **Matlab Function**	**Continuous definition**	**Discrete definition**
L1 norm ℓ_1 `norm(·,1)`	$\|x\|_1 = \int_{t\in\Omega} \|x(t)\| dt$	$\|x\|_1 = \sum_{n=1}^{N} \|x_n\|$
L2 norm ℓ_2 `norm(·)`	$\sqrt{\|x\|_2 = \int_{t\in\Omega} \|x(t)\|^2 dt}$	$\|x\|_2 = \sqrt{\sum_{n=1}^{N} \|x_n\|^2}$
L12 mixed norm ℓ_{12} `norm_L12(·,g_d,g_t)`	$\|x\|_{12} = \int_{t_1\in\Omega_1} \left(\int_{t_2\in\Omega_2} \|x(t_1,t_2)\|^2 dt_2 \right)^{\frac{1}{2}} dt_1$	$\|x\|_{12} = \sum_{g\in G} \|x_g\|_2,$ with $x_g \subset x$
L1∞ mixed norm $\ell_{1\infty}$ `norm_L1inf(·,g_d,g_t)`	$\|x\|_{1\infty} = \int_{t_1\in\Omega_1} \left(\max_{t_2\in\Omega_2} \|x(t_1,t_2)\| \right) dt_1$	$\|x\|_{1\infty} = \sum_{g\in G} \|x_g\|_\infty,$ with $x_g \subset x$
TV norm ℓ_{TV} `TV_norm(·)` or `TV_norm3D(·)`	$\|x\|_{TV} = \int_{t\in\Omega} \|\nabla x(t)\| dt$	$\|x\|_{TV} = \sum_{n} \|\nabla x(n)\|$
Nuclear norm ℓ_* `norm_nuclear(·)`		For x a m by$n$ matrix with singular values σ_i, we have: $\|x\|_* = \sum_{i=1}^{\min\{m,n\}} \sigma_i$

Table 2: Presentation of the norms

5.2 Norm proximal operators

Minimizing norm presented in table 2 leads to common proximal operators. We present them in this section. All Matlab prox functions take three parameters: `x, lambda, param`. First, `x` is the initial signal. Then `lambda` is the weight of the objective function. Finally, `param` is a Matlab structure that containing a set of optional parameters.

- `param.max_iter`: The maximum number of iterations (default 200).
- `param.epsilon`: This parameter is used to define the stopping criterion of the problem. The algorithm stops if
$$\frac{f(t) - f(t-1)}{f(t)} < \epsilon$$
where $f(t)$ is the function to be minimized at iteration t. $\epsilon \in \mathbb{R}_+^*$ (default 10^{-4}.)
- `param.verbose`: Log parameter: 0 no log, 1 a summary at convergence, 2 print main steps (default 1.)

Other specific parameters are described for each prox function.

5.2.1 ℓ_1 norm proximal operator

The ℓ_1 norm proximal operator solves the problem.

$$\text{prox}_{\lambda\|\Psi\cdot\|_1}(x) = \min_z \frac{1}{2}\|x - z\|_2^2 + \lambda\|\Psi(z)\|_1.$$

The associated Matlab function `prox_L1` is:

```
function sol = prox_L1(x, lambda, param).
```

The following fields of `param` are specific to the `prox_L1` function:
- `param.Psit`: Operator Ψ (default: Id).
- `param.Psi`: Adjoint of Ψ: Ψ^* (default: Id).
- `param.tight`: 1 if Ψ is a tight frame or 0 if not (default = 1).
- `param.nu`: ν is a bound on the norm of the operator Ψ, i.e. $\|\Psi x\|^2 \leqslant \nu\|x\|^2$ (default: 1).
- `param.weights`: weights for a weighted L1-norm (default = 1).

5.2.2 ℓ_{12} norm proximal operator

The ℓ_{12} norm proximal operator solves the problem.

$$\text{prox}_{\lambda\|\cdot\|_{12}}(x) = \min_z \frac{1}{2}\|x - z\|_2^2 + \lambda\|z\|_{12}.$$

The associated Matlab function `prox_L12` is:

```
function sol = prox_L12(x, lambda, param).
```

The following fields of `param` are specific to the `prox_L12` function:
- `param.g_d`: contains the indices of the elements to be grouped (row vector).
- `param.g_t`: contains the size of the different group(row vector).
- `param.g_b`: just call back_perm(g_d). This argument is optional, but it may accelerate the algorithm.
- `parm.multi_group`: in order to use group with overlap, activate the multi_group option (set it to 1) (default 0.)
- `param.weights2`: weights for a weighted ℓ_{12} norm works on the norm ℓ_2 (default = ones.)
- `param.weights1`: weights for a weighted ℓ_{12} norm works on the norm ℓ_1 (default = ones.)

Example: `param.g_d` and `param.g_t` Suppose we have $x = [x_1, x_2, x_3, x_4, x_5, x_6]$ we want to group $[x_1, x_2, x_4, x_5]$ and $[x_3, x_6]$. Then we set:
`param.g_d=[1 2 4 5 3 6]'` and `param.g_t=[4 2]'`
It is also possible to set:
`param.g_d=[4 5 3 6 1 2]'` and `param.g_t=[2 4]'`.

Overlaping group: In order to make overlapping group `param.g_d`, `param.g_t` must be vectors of non overlapping group. Example:
```
param.g_d=[g_d1, g_d2, ..., g_dn];
param.g_t=[g_t1, g_t2, ..., g_tn];
```
There must be no overlap in `g_d1, g_d2,... g_dn`.

5.2.3 $\ell_{1\infty}$ norm proximal operator

The $\ell_{1\infty}$ norm proximal operator solves the problem.

$$\text{prox}_{\lambda\|\cdot\|_{12}}(x) = \min_z \frac{1}{2}\|x-z\|_2^2 + \lambda\|z\|_{1\infty}.$$

The associated Matlab function `prox_L1inf` is:

```
function sol = prox_L1inf(x, lambda, param).
```

The following fields of `param` are specific to the `prox_L1inf` function:
- `param.g_d`: contains the indices of the elements to be grouped (row vector).
- `param.g_t`: contains the size of the different group(row vector).
- `param.g_b`: just call back_perm(g_d). This argument is optional, but it may accelerate the algorithm.
- `parm.multi_group`: in order to use group with overlap, activate the multi_group option (set it to 1) (default 0.)
- `param.weights2`: weights for a weighted ℓ_{12} norm works on the norm ℓ_2 (default = ones.)
- `param.weights1`: weights for a weighted ℓ_{12} norm works on the norm ℓ_1 (default = ones.)

Example: `param.g_d` and `param.g_t` Suppose we have $x = [x_1, x_2, x_3, x_4, x_5, x_6]$ we want to group $[x_1, x_2, x_4, x_5]$ and $[x_3, x_6]$. Then we set:
`param.g_d=[1 2 4 5 3 6]'` and `param.g_t=[4 2]'`
It is also possible to set:
`param.g_d=[4 5 3 6 1 2]'` and `param.g_t=[2 4]'`.

Overlaping group: In order to make overlapping group `param.g_d`, `param.g_t` must be vectors of non overlapping group. Example:
```
param.g_d=[g_d1, g_d2, ..., g_dn];
param.g_t=[g_t1, g_t2, ..., g_tn];
```
There must be no overlap in `g_d1, g_d2,... g_dn`.

5.2.4 TV norm proximal operator

The ℓ_{TV} norm proximal operator solves the problem.

$$\text{prox}_{\lambda\|\cdot\|_{TV}}(x) = \min_z \frac{1}{2}\|x-z\|_2^2 + \lambda\|z\|_{TV}.$$

The associated Matlab function `prox_TV` is:

```
function sol = prox_TV(x, lambda, param),
```

where x is a 1 or 2 dimensions matrix.

5.2.5 3D TV norm proximal operator

The ℓ_{3DTV} norm proximal operator solves the problem.

$$\text{prox}_{\lambda\|\cdot\|_{TV}}(x) = \min_z \frac{1}{2}\|x-z\|_2^2 + \lambda\|z\|_{TV}.$$

The associated Matlab function `prox_TV3D` is:

```
function sol = prox_TV3D(x, lambda, param),
```

where x is a 1,2 or 3 dimensions matrix.

5.2.6 Nuclear norm proximal operator

The ℓ_* norm proximal operator solves the problem.

$$\text{prox}_{\lambda\|\cdot\|_*}(x) = \min_z \frac{1}{2}\|x-z\|_2^2 + \lambda\|z\|_*.$$

The associated Matlab function `prox_NuclearNorm` is:

```
function sol = prox_NuclearNorm(x, lambda, param).
```

5.3 Projection operator

5.3.1 Projection on B1-ball

The B1-Ball projection operator solves the problem.

$$\min_z ||x-z||_2^2 \text{s.t.} ||w \cdot z||_1 < epsilon$$

The associated Matlab function `proj_B1` is:

```
function sol = proj_B1(x, lambda, param).
```

`x` is the vector be projected. `lambda` is compatibility parameter. It is not used in the function. The following fields of `param` are specific to the `proj_B1` function:

- `param.w`: contain the weigths (default ones.)
- `param.epsilon`: Radius of the L1 ball (default = 1e-3.)
- `param.verbose`: Log parameter: 0 no log, 1 a summary at convergence, 2 print main steps (default 1.)

5.3.2 Projection on B2-ball

The B2-Ball projection operator solves the problem.

$$\min_z ||x-z||_2^2 \text{s.t.} ||y-Az||_2 < epsilon$$

The associated Matlab function `proj_B2` is:

```
function sol = proj_B2(x, lambda, param).
```

`x` is the vector be projected. `lambda` is compatibility parameter. It is not used in the function. The following fields of `param` are specific to the `proj_B1` function:

- `param.A`: Operator A (default: Id).
- `param.At`: Adjoint of A: A^* (default: Id).
- `param.tight`: 1 if Ψ is a tight frame or 0 if not (default = 1).
- `param.nu`: ν is a bound on the norm of the operator Ψ, i.e. $\|\Psi x\|^2 \leqslant \nu\|x\|^2$ (default: 1).
- `param.epsilon`: Radius of the L2 ball (default = 1e-3.)
- `param.max_iter`: The maximum number of iterations (default 200).
- `param.tol`: tolerance for the projection onto the L2 ball. The algorithms stops if

$$\frac{\epsilon}{1-tol} \leqslant \|y-Az\|_2 \leqslant \frac{\epsilon}{1+tol}$$

  (default: 1e-3).
- `param.verbose`: Log parameter: 0 no log, 1 a summary at convergence, 2 print main steps (default 1.)

5.4 Sum of proximal operator

This function allows the user to solve the proximal operator of K functions:

$$\text{prox}_{\lambda \sum g}(x) = \min_z \frac{1}{2}\|x - z\|_2^2 + \lambda \sum_{n=1}^{K} w_n g_n(z).$$

The associated Matlab function `prox_sumG` takes three parameters

```
sol = prox_sumG(x, lambda , param).
```

`param` is a Matlab structure that containing a set of parameters. The following two fields are specific to the `prox_sumG` function:

- `param.w`: w the weight (all equal by default). Note that $\sum_{i=1}^{K} \text{param.W}(i) = 1$.
- `G`: is an array of structures representing the functions $g_1, g_2, ..., g_K$. Each of these structures needs at least two fields: `G(i).norm` and `G(i).prox` with $i \in \{1, 2, ..., K\}$. The former is a Matlab function that takes as input a vector $x \in \mathbb{R}^N$ and returns the value $f_i(x)$, the latter is a Matlab function that takes as input a vector $\in \mathbb{R}^N$, a strictly positiv real number τ and returns the vector $\text{prox}_{\tau g_i}(x)$ (In Matlab, write: `G(i).prox=@(x, T) prox_gi(x, T)`, where `prox_gi(x, T)` solves the problem $\text{prox}_{Tg_i}(x)$ given in equation 3).
- `param.gamma`: step-size parameter γ for the generalized forward backward algorithm. This constant should satisfy: $\gamma \in [\epsilon, 2/\beta - \epsilon]$, for $\epsilon \in]0, 1[$.
- `param.epsilon`: This parameter is used to define the stopping criterion of the problem. The algorithm stops if
$$\frac{f(t) - f(t-1)}{f(t)} < \epsilon$$
where $f(t)$ is the function to be minimized at iteration t. $\epsilon \in \mathbb{R}_+^*$ (default 10^{-3}.)
- `param.max_iter`: The maximum number of iterations (default 100).
- `param.verbose`: Log parameter: 0 no log, 1 a summary at convergence, 2 print main steps (default 1.)
- `param.lambda_t`: λ_t weight of the update term $\in [\epsilon, 1]$ for the generalized forward backward algorithm. By default, it's equal to one.

Remark: The generalized forward backward solver is used to solve this problem.

6 Examples

In this Section, we illustrate the use of the toolbox with two different examples. All the proximal operators used for these are already implemented in the toolbox.

6.1 Example 1: inpainting

The goal is to recover some missing pixels in an image. The pixels' values form a $M \times M$ matrix. Let's define a function $t(X)$ that transform a $M \times M$ matrix into a row vector $\boldsymbol{x}$ of size M^2. The inverse function will be written $t^{-1}(\boldsymbol{x})$. Suppose that the original image is I^* and the missing pixel are represented by 0 in mask M_I (the other values are 1).The number of known pixels is K. The measurements are simply

$$\boldsymbol{b} = t(M_I \bullet I) = t(M_I) \bullet t(I) = \boldsymbol{m}_i \bullet \boldsymbol{i} = \mathsf{A}\boldsymbol{i},$$

with $(\cdot \bullet \cdot)$ the element by element product and A a $K \times M^2$ binary matrix with only one single 1 value per line where the pixels are knows.

The image is recovered using a Total Variation prior (The L1 norm of the magnitude of the gradient, for details see [13]). The problem can be formulated

$$\boldsymbol{s} = \min_{\boldsymbol{x}} \|t^{-1}(\boldsymbol{x})\|_{TV}, \quad s.t \quad \|\boldsymbol{b} - \mathsf{A}\boldsymbol{x}\|_2 \leqslant \epsilon, \tag{13}$$

with $x \in \mathbb{R}^{M^2}$. The reconstructed image is $I = f^{-1}(s)$.

This problem is solved with DR algorithm. To apply this algorithm, 13 is split into two sub-functions f_1 and f_2.

f_1 is the TV norm

$$f_1 = \|t^{-1}(\boldsymbol{x})\|_{TV} = \|X\|_{TV}. \tag{14}$$

f_2 is the indicator function of the convex set $S = \{\boldsymbol{x} : \|\mathsf{A}\boldsymbol{x} - \boldsymbol{b}\| \leqslant \epsilon\}$ (i.e. B2-ball of radius ϵ centred in $\boldsymbol{b}$). It is defined by

$$f_2(\boldsymbol{x}) = i_{c2}(\boldsymbol{x}) = \begin{cases} 0, & \text{if } \|\mathsf{A}\boldsymbol{x} - \boldsymbol{b}\|_2 \leqslant \epsilon \\ \infty, & \text{otherwise.} \end{cases} \tag{15}$$

Note that the proximal operator of f_2 is a projection onto the set S.

The solution of the problem is shown on figure 1. See the code "Example_douglas_rachford.m" for more details.

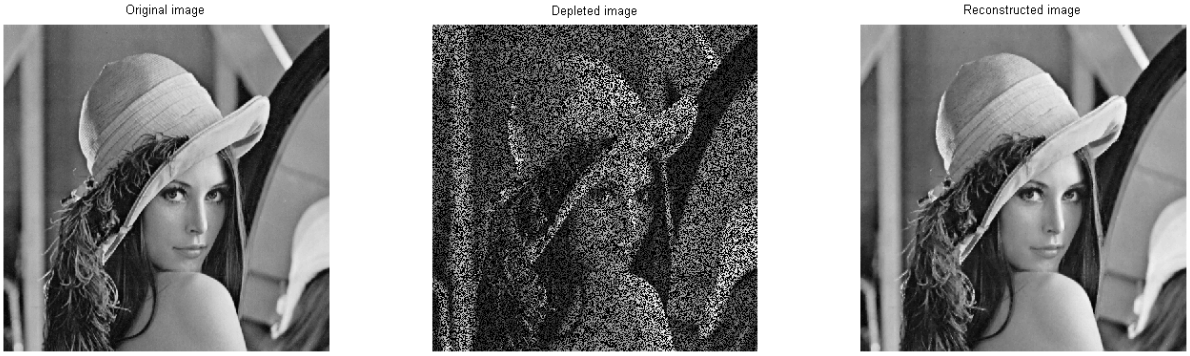

Figure 1: Inpainting demonstration with Douglas-Rachford. 50 percent of the pixels have been removed. $\epsilon = 10^{-1}$

6.2 Example 2: denoising

Here the goal is to remove a Gaussian noise on the image. Tree assumptions have been made. The image is not very different from the measurement. It has a small TV-norm and its Haar wavelet's decomposition is sparse.

Definition are the same as in example 1 with a difference: $\boldsymbol{b} = t(I^* + G_n(\sigma))$ with $G_n(\sigma)$ a $M \times M$ matrix made of σ variance random Gaussian noise.

The problem can be expressed with this equation:

$$\boldsymbol{s} = \min_{\boldsymbol{x}} \|\boldsymbol{x} - \boldsymbol{b}\|^2 + \tau_1 \cdot \|X\|_{TV} + \tau_2 \cdot \|t(H(X))\|_1, \tag{16}$$

where $H(Image)$ decompose the image in Haar wavelet and $X = t^{-1}(\boldsymbol{x})$. The denoised image is $I = f^{-1}(s)$.

As there are three functions in this problem, it's not possible to apply directly the forward-backward algorithm which require only two functions. But, this problem can be solve with a generalized forward-backward algorithm [**?**].

Two functions are be defined: $g_1 = \tau_1 \cdot \|X\|_{TV}$ and $g_2 = \tau_2 \cdot \|H(X)\|_1$.

Solving (16) is equivalent as searching the solution of the proximal operator on $(g_1 + g_2)(\boldsymbol{x})$. The function sum_proxG in the toolbox find the solution of the proximal operator of a sum of function. Thus it is called to solve (16).

The solution is shown on figure 2. For more details see "Example_prox_multi_function.m".

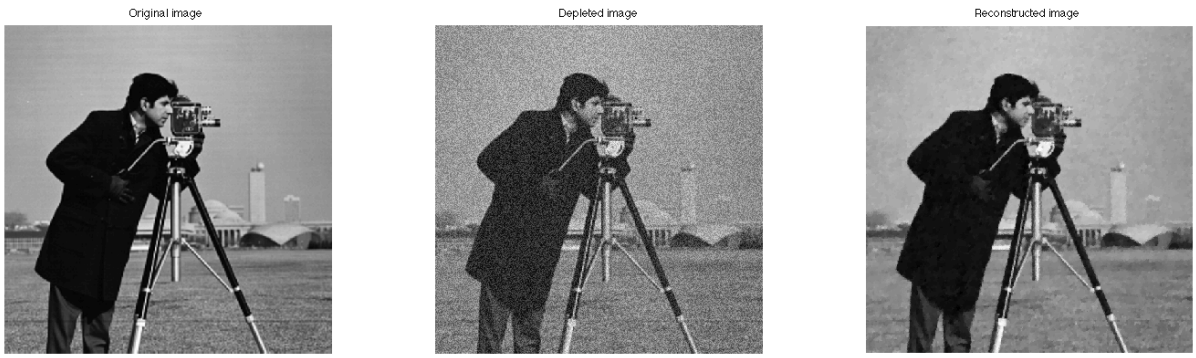

Figure 2: Denoising demonstration with generalized forward-backward splitting. The noise is Gaussian with $\sigma = 10$. $\tau_1 = 20$ et $\tau_2 = 3$

7 Summary of all functions

Function	Explanation	Argument
douglas_rachford	This function solves the optimization problem of minimizing the sum of f_1 and f_2 using the Douglas-Rachford algorithm. Both functions have to be convex. $$\min_x f_1(x) + f_2(x)$$	x0 ,f1, f2, param
forward_backward	This function solves the optimization problem of minimizing the sum of f_1 and f_2 using the forward-backward algorithm. Both functions have to be convex and one needs to be β-Lipschitz. $$\min_x f_1(x) + f_2(x)$$	x0, f1, f2, param
admm	This function solves the optimization problem of minimizing the sum of f_1 and f_2 using the alternating-direction method of multipliers (ADMM) algorithm. Both functions have to be convex. $$\min_x (f_1(x) + f_2(y)) \quad \text{such that} \quad y = Lx$$	x0,f1, f2, L, param
generalized_forward_backward	This function solves the optimization problem of minimizing the sum of F_i using the generalized forward-backward algorithm. All functions need to be convex and one needs to be β-Lipschitz (denoted f.) $$\min_x f(x) + \sum_i w_i F_i(x)$$	x0, F, f , param
ppxa	This function solves the optimization problem of minimizing the sum of F_i using the parallel proximal algorithm (PPXA) which is a generalization of the forward backward algorithm. All functions have to be convex. $$\min_x \sum_i w_i F_i(x)$$	x0, F, param
sdmm	This function solves the optimization problem of minimizing the sum of F_i using the simultaneous-direction method of multipliers (SDMM) algorithm. All functions need to be convex. $$\min_x \sum_i F_i(L_i x)$$	F, param
RLR	This function solves a common optimisation problem: $$\min_x \|Ax - x_0\|_2^2 + f(x)$$	x0, f, A, At, param

Table 3: List of main functions

Function	Explanation	Argument
prox_L1	ℓ_1 norm proximal operator. Solve: $$\min_x \frac{1}{2}\|x-z\|_2^2 + \lambda\|\Psi(x)\|_1$$	x, lambda, param
Prox_Linf	ℓ_∞ norm proximal operator. Solve: $$\min_x \frac{1}{2}\|x-z\|_2^2 + \lambda\|\Psi(x)\|_\infty$$	x, lambda, param
prox_TV	TV norm proximal operator. Solve: $$\min_x \frac{1}{2}\|x-z\|_2^2 + \lambda\|x\|_{TV}$$	x, lambda, param
prox_TV3D	TV norm proximal operator in 3 dimension. Solve: $$\min_x \frac{1}{2}\|x-z\|_2^2 + \lambda\|x\|_{TV}$$	x, lambda, param
prox_L12	ℓ_{12} norm [2] proximal operator. Solve: $$\min_x \frac{1}{2}\|x-z\|_2^2 + \lambda\|x\|_{12}$$	x, lambda, param
prox_L1inf	$\ell_{1\infty}$ norm [3] proximal operator. Solve: $$\min_x \frac{1}{2}\|x-z\|_2^2 + \lambda\|x\|_{1\infty}$$	x, lambda, param
prox_Nuclear_Norm	ℓ_* norm proximal operator. Solve: $$\min_x \frac{1}{2}\|x-z\|_2^2 + \lambda\|x\|_*$$	x, lambda, param
ProjB1	Projection on B1-ball. lambda is an unused parameter for compatibility reasons. Solve: $$\min_x \|x-z\|_2^2 \quad s.t. \quad \|x\|_1 < \epsilon$$	x, lambda, param
ProjB2	Projection on B2-ball. lambda is an unused parameter for compatibility reasons. Solve: $$\min_x \|x-z\|_2^2 \quad s.t. \quad \|x\|_2 < \epsilon$$	x, lambda, param
sum_proxG	Compute the proximal operator of a sum of functions Solve: $$\min_x 0.5 * \|x-z\|_2^2 + \lambda \cdot \sum_i G_i(x)$$	x, lambda, param

Table 4: List of proximal operators

Function	**Explanation**	**Argument**
normL12	This function compute the ℓ_{12} norm of x. The group ensemble is denoted G. $$\|x\|_{12} = \sum_{g \in G} \|x_g\|_2$$	x, g_d, g_t
normL1inf	This function compute the $\ell_{1\infty}$ norm of x. The group ensemble is denoted G. $$\|x\|_{1\infty} = \sum_{g \in G} \|x_g\|_\infty$$	x, g_d, g_t
TV_norm	This function compute the TV norm of x. $$\|x\|_{TV}$$	x
norm_nuclear	This function compute the nuclear norm of x. $$\|x\|_*$$	x

Table 5: List of norm functions

References

[1] D. Gabay, "Applications of the method of multipliers to variational inequalities," in *Augmented Lagrangian Methods: Applications to the Numerical Solution of Boundary-Value Problems*, M. Fortin and R. Glowinski, Eds. Amsterdam: North Holland, 1983, pp. 299–331.

[2] P. Tseng, "Applications of a splitting algorithm to decomposition in convex programming and variational inequalities," *SIAM J. Control Optim.*, vol. 29, pp. 119–138, 1991.

[3] P. L. Combettes and V. R. Wajs, "Signal recovery by proximal forward-backward splitting," *Multiscale Modeling and Simulation*, vol. 4, no. 4, pp. 1168–1200, Nov. 2005.

[4] J. Douglas and H. H. Rachford, "On the numerical solution of heat conduction problems in two or three space variables," *Trans. Amer. Math. Soc.*, vol. 82, pp. 421–439, 1956.

[5] P. L. Lions and B. Mercier, "Splitting algorithms for the sum of two nonlinear operators," *SIAM J. Numer. Anal.*, vol. 16, pp. 964–979, 1979.

[6] J. Eckstein and D. P. Bertsekas, "On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators," *Math. Programming*, vol. 55, pp. 293–318, 1992.

[7] P. L. Combettes, "Solving monotone inclusions via compositions of nonexpansive averaged operators," *Optimization*, vol. 53, pp. 475–504, 2004.

[8] P. L. Combettes and J. C. Pesquet, "A Douglas-Rachford splitting approach to nonsmooth convex variational signal recovery," *IEEE J. Selected Topics Signal Process.*, vol. 1, pp. 564–574, 2007.

[9] B. Martinet, "Determination approchée d'un point fixe d'une application pseudo-contractante. Cas de l'application prox," *Comptes Rendus de l'Academie des Sciences, Paris, Série A*, vol. 274, pp. 163–165, 1972.

[10] R. Rockafellar, "Monotone operators and the proximal point algorithm," *SIAM J. Control Optim.*, vol. 14, pp. 877–898, 1976.

[11] P. L. Combettes and J.-C. Pesquet, "Proximal splitting methods in signal processing," in *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*, H. H. Bauschke, R. Burachik, P. L. Combettes, V. Elser, D. R. Luke, and H. Wolkowicz, Eds. Springer-Verlag, 2011, pp. 185–212.

[12] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM J. Img. Sci.*, vol. 2, pp. 183–202, March 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1658360.1658364

[13] A. Chambolle, "An algorithm for total variation minimization and applications," *J. Math. Imaging Vis.*, vol. 20, pp. 89–97, Jan. 2004.